**Assumption: No external libraries are used in the code to support big numbers as the same was confirmed with the TA. Hence, have went with u\_int64\_t which is available as a part of standard c installation.**

**Results:**

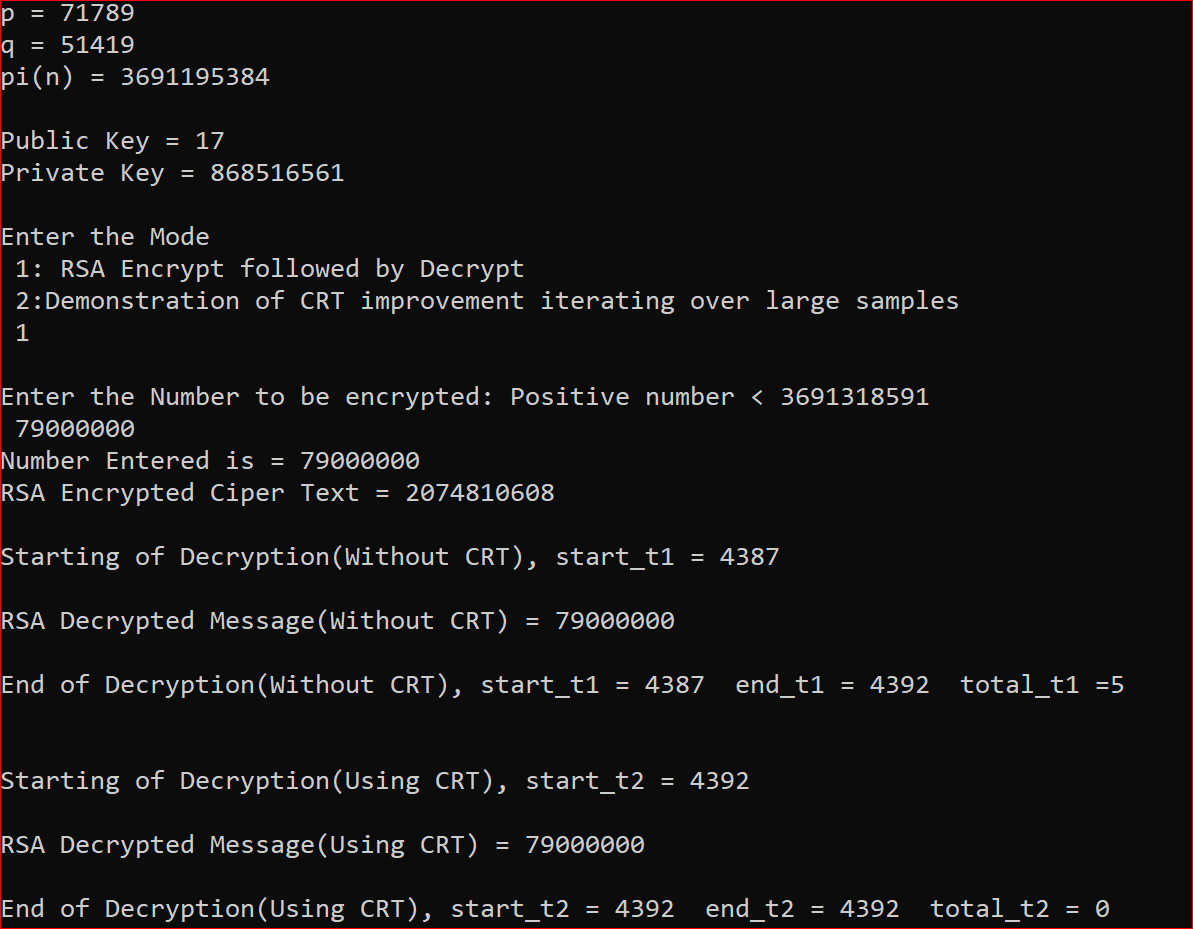
CRT has been demonstrated to show improvements in decryption due to processing of modulo P/Q numbers which has smaller magnitude as opposed to direct modulo N processing as captured in **Sample Output:** As the digits being processed are less than 1024 bits, in some cases, especially for smaller numbers, sometimes CRT was taking more time, probably as it would requires 2 modulo operation with P and Q and additional multiplication and addition as opposed to 1 modulo exponentiation operation in the direct decryption. It was also observed that the improvements increase as P, Q and input Number increases in magnitude. It was noticed that Max improvement observed is greater when the code is ran in Linux machines as compared to windows. This could be because the clock function measures wall-time in Windows and CPU time in Linux.

The improvement would have been much higher, if the 1024-bit N are used but it is avoided as it requires installation of specialized big number libraries such as GMP which may require additional approvals for installation on office PC and added to that also TA would need similar installations to replicate the observations which makes it outside the scope of the assignment.

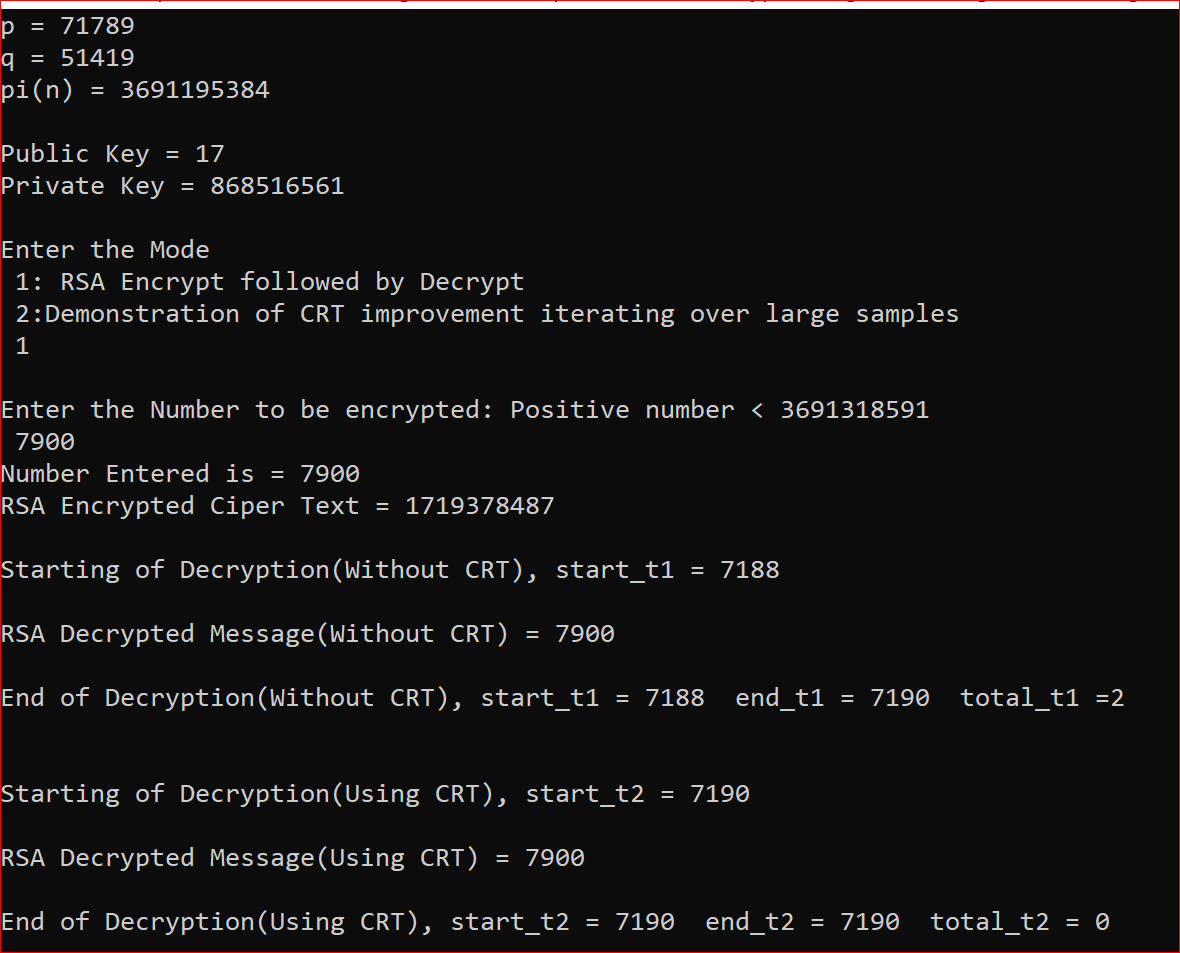
**Sample Output: Mode 1**

Note: Mode1 results as observed on Windows System

**A) Total Decryption time(Total\_t1) = 5 units without CRT Vs Total\_t1 = 0 units with CRT**

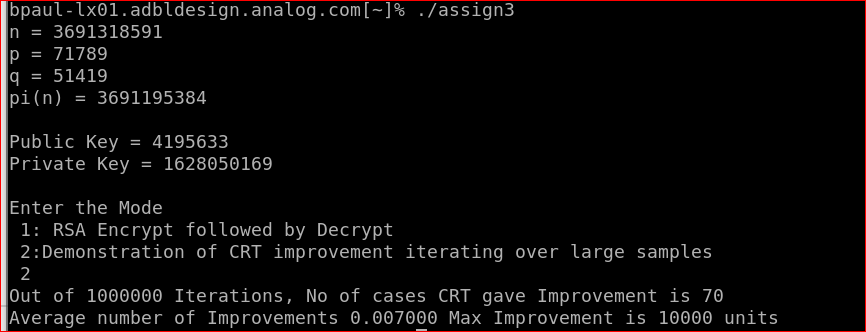


**B) Total Decryption time(Total\_t1) = 2 units without CRT Vs Total\_t1 = 0 units with CRT**



**Sample Output: Mode 2**

**A) Max Improvement of 10,000 units observed when cycling through large number of inputs(As Observed on Linux)**



**B) Number of cases which CRT gave improvement is about 420(As Observed in Windows)**

